PatchCity

PatchCity: Procedural City Generation using Texture Synthesis

John D. Bustard* and Liam P. de Valmency**

*Queen's University Belfast, UK
**University of Southampton, UK

Abstract

PatchCity is a new approach to the procedural generation of city models. The algorithm uses texture synthesis to create a city layout in the visual style of one or more input examples. Data is provided in vector graphic form from either real or synthetic city definitions. The paper describes the PatchCity algorithm, illustrates its use, and identifies its strengths and limitations. The technique provides a greater range of features and styles of city layout than existing generative methods, thereby achieving results that are more realistic. An open source implementation of the algorithm is available.

Keywords: procedural modeling, texture synthesis, example-based

1 Introduction

Procedural modeling techniques have substantially reduced the effort required to create synthesised environments. In the computer games industry, for example, sandbox games [Sony Online Entertainment, 2014] can now be set in vast explorable worlds, and the player presented with an ever-changing environment each time they enter the game. Similarly, in film, generative modeling has greatly simplified the creation of wide environment shots, backdrops, and the in-fill of detailed scenery [Planetside Software, 2014] [IDV Inc., 2014].

Existing procedural techniques for city generation use a variety of synthesis methods to produce road layouts and building placements. These include manually created heuristics [Parish and Müller, 2001], agent models [Lechner et al., 2003], tensor fields [Chen et al., 2008] and statistically based constraints [Groenewegen et al., 2009]. Such techniques can produce realistic approximations of cities but have limitations. In particular:

- The structure of a created city is an emergent property of the underlying heuristic rules that determine its shape. The variety of cities produced is therefore constrained by the creative power of those heuristics. As a result, existing procedural city techniques are often restricted to grid-based, metropolitan layouts.
- The fine tuning of a created city's appearance is achieved through adjusting the parameters of the generative algorithm. As the heuristics become increasingly complex, the tuning of the associated parameters also increases in difficulty [Lechner et al., 2003].
- When complex heuristics and constraints are applied, the time needed to generate structures can increase significantly. This can reach a point where construction can no longer occur in real-time, relying instead on offline generation; this may not be suitable for some applications.

This paper describes a texture synthesis approach to city generation, PatchCity, which was conceived as a way of addressing these limitations. The strategy is to extrapolate from data provided in the form of vector graphic city layouts to create a new, larger map, which maintains the consistency and visual style of the original. The advantages of this approach are:

- The variation in city styles and features is not constrained by the algorithm because they are produced directly from the patterns in the user-supplied input.
- By adjusting the input data, subtle changes in city appearance can be achieved in an intuitive way. In addition, texture synthesis methods can also be applied to fill holes around explicitly defined city features, enabling precise adjustment of the city structure as required.
- Because the method uses a look-up table of city patches, the city structure can be produced efficiently without requiring complex heuristic rules to create detailed results. In addition, the use of existing data sources can also handle building and other feature placement implicitly, which requires significant additional processing within other city generation algorithms.

Section 2 of the paper examines existing work in procedural city generation. Section 3 then describes the PatchCity algorithm and related work in texture synthesis. The section shows how vector-based texture synthesis can be performed using urban layouts and identifies the adjustment steps that are applied to improve the resulting structure. Section 4 presents the results of applying the PatchCity algorithm to four example urban locations. The section highlights the strengths and limitations of the algorithm. The paper concludes with a discussion of potential future work.

2 Procedural City Generation

CityEngine, designed by Parish and Müller [Parish and Müller, 2001] was the first algorithm developed for automatically constructing city road networks. CityEngine takes road network descriptions as input, and creates a skeleton city layout. The heuristic rules for the city's construction are expressed as a set of Lindenmayer System (L-system) formal grammars [Rozenberg and Salomaa, 1980]. During creation, consistency adjustments are also made to ensure, for example, that roads do not overlap. The heuristics and constraints require additional user input in the form of images representing water boundaries, terrain elevation, population density and street patterns. The use of standard highway patterns, such as grid-aligned, radial, and population-density seeking, helps give the generated city a realistic appearance.

CityEngine is effective in situations where a "good enough" output is acceptable, such as stylised video games. One problem with this approach, however, is that the indirect nature of L-system descriptions makes them difficult to adjust, especially as they increase in size to deal with detailed map features. Lechner et al. [Lechner et al., 2003] proposed an alternative agent-based approach that helps overcome these issues. Their technique delegates the handling of road and building placement to agents that follow specific rules, similar to those used by city planners. Different agent types are assigned specific tasks, such as extending roads into unexplored terrain, connecting existing roads to create shortcuts, and searching for available land on which to construct buildings. Building construction is based on the type of zone being built and the estimated land value.

Lechner's approach requires less user input than CityEngine, needing only one image. It also has the advantage of allowing the user to set global parameters to fine tune the appearance of the generated city. In later work, the authors combined their system with the SimCity engine for more aesthetically pleasing output [Lechner et al., 2004]. There are, however, two limitations with an agent-based approach. One is that basing generation on the emergent results of a set of heuristics limits the control over the resulting city appearance. The second is that generation speed is relatively slow.

Groenewegen et al. [Groenewegen et al., 2009] introduced a strategy for using real-world exemplar data in the generation of cities. The models govern how a city might be formed based on user-specified factors such as city size, continent, historical background and number of highways passing through it. The generation process divides a city into zones which represent various types of buildings, including light or heavy industrial zones, commercial districts and transport nodes. These zones are grouped according to their statistical likelihood of adjacency, based on the input parameters. For example, a city in Western Europe would have a different zone grouping to one in North America. The Groenewegen approach has the obvious overhead of requiring a new land use model for each specific country in which a city is generated, which implies having an understanding of

building patterns in that situation. Also, it focuses on the high-level structure for a city, meaning that additional work is needed to provide the detail in each case.

Another strategy for generating road networks from pre-existing data is described by Aliaga et al. [Aliaga et al., 2008], where new roads are produced by analysing statistical properties of an existing network. Their system uses a random walk algorithm, which iteratively adds road segments to the existing network in a way that mimics the properties of road intersection points in the original, such as mean/variance of road distances and angles, tortuosity and intersection hierarchy levels. As a result, the road network generated is one of the most accurate among those available. Its implementation is, however, much more complex than other approaches. Also, it can be biased by highly atypical streets from an example map, which may skew the intersection property calculations. Texture warping is used to fill space between roads to create convincing satellite imagery.

Tackling the difficulty of indirect adjustment of city descriptions, as required in CityEngine, Chen et al. [Chen et al., 2008] proposed an alternative interactive approach. This allows users to create a road network from scratch or modify an existing network. The underlying representation is a tensor field which can be manipulated directly or modified through the graph representing the resulting road network. The creation engine requires the user to know the general layout of the road network they wish to generate, which prevents creation of a random but realistic city derived from common city patterns. This becomes an advantage, however, if the user has a very specific city layout in mind. The key strength of the method is the realism of the output, which improves on that of CityEngine.

From this analysis, five desirable properties of a city generation algorithm are evident: realism, detail, automation, control and speed of execution. Each existing approach varies in the degree to which these properties are achieved. The next section presents an algorithm intended to improve on the detail and realism of existing results. The approach can be fully automatic and has the potential for low level adjustment through explicit patch placement and hole filling. It also operates sufficiently quickly for interactive applications.

3 PatchCity Approach

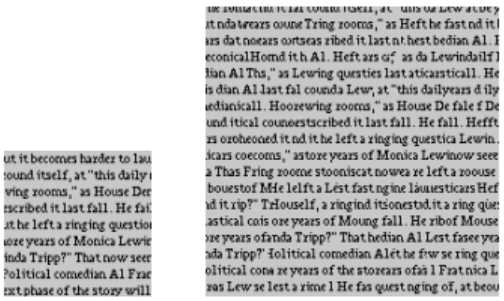


Figure 1: Results of applying texture synthesis to an image of text.

PatchCity generates new city structure through a technique based on texture synthesis. In general, texture synthesis algorithms aim to extrapolate from data provided in the form of an image to create a new, larger image which maintains the consistency and visual style of the original. Figure 1, for example, shows the technique applied to a block of text. The visual appearance of the generated block on the right strongly resembles that of the initial sample patch on the left from which it was produced. Note that no semantic information has been used in the synthesis process so the generated text is meaningless.

PatchCity is based on a similar synthesis approach but uses patches in vector graphic form that include important semantic information. More specifically, it uses patches in the form of maps from Geographic Information Systems (GIS) such as OpenStreetMap [Haklay and Weber, 2008]. Its approach means that it can handle building and other city feature placement, which previous city generation algorithms have had to process separately.

Current algorithms for texture synthesis largely fall into two categories: pixel-based [Efros and Leung, 1999] and patch-based methods [Efros and Freeman, 2001] [Kwatra et al., 2003]. One issue with per-pixel synthesis methods, such as non-parametric pixel sampling [Efros and Leung, 1999], is that they fail to preserve the high-level structure of road networks, resulting in disorganised clusters of road segments. Also, existing patch-based methods, such as Image Quilting or Graphcut, create frequent repetition of certain sampled patches, and have road structure disconnection along patch seams. PatchCity addresses these issues by performing texture synthesis using a vector representation. This format enables patch constraints to be specified more precisely and provides a representation that enables patches to be merged without noticeable errors.

3.1 Algorithm

The PatchCity algorithm takes one or more vector street maps as input, from which patches can be extracted. For image based texture synthesis, sampling of the original image(s) can take place in real-time, as patches can be calculated entirely using local pixel values. In a vector format, this process becomes more difficult, as roads will be represented as a series of points marking their path, and buildings by a set of points denoting their shape. To obtain the set of road points and buildings which lie within a patch, at a given point, requires clipping of path segments, as well as exclusion of buildings which do not fall entirely within the patch area. The first step in the algorithm is to divide each input map into its respective patches, and to store these in a lookup table.

Patch extraction takes place at a user-specified interval along each axis of the 2D image. Each extracted patch stores the paths which cross through its area, clipped to the boundaries of the map region from which it was taken. The Cohen-Sutherland algorithm [Sproull and Newman, 1973] is used to clip any lines that pass through the patch, and store them. Buildings that fall entirely within the patch are also stored.

To prevent repeated iteration through each path node within a patch when locating nodes that lie on its boundary, results are cached for further use. This includes the road count along each edge, the road locations (as a distance from the edge's top or leftmost point), and the road sizes. The use of the cache improves the efficiency of the algorithm and also enables boundary values to be compared more conveniently. Specifically, execution time is reduced because only those patches with equivalent road counts along the patch boundary need to be considered as candidates.

Map generation takes place in a top to bottom, left to right fashion, following a grid of patch-sized areas. In addition to the patch extraction interval, the user can also specify the width and height of the synthesised map (the number of patches along each axis), as well as the size of each of the patches themselves. For each unfilled space in the map, the algorithm will fetch the most appropriate patch from the lookup table.

While regular texture synthesis algorithms perform a nearest neighbour comparison with pixel values, the PatchCity algorithm takes a different approach. Firstly, the top-left patch in the map is initialised to a random fragment from the lookup table containing at least one road. Subsequent patches are chosen based on a cost function which takes two patches as arguments. In choosing a new patch from the lookup table, only those with an equivalent number of roads along the connecting edge are evaluated. The cost function must be calculated for each edge which connects the target patch location to those which have already been placed in the synthesised map. For patches in either the first row or the first column, only one evaluation takes place (against the patch to the left and the patch above respectively). For each of the remaining patches, the choice of patch is based on the sum of the cost function run against both adjacent patches. The algorithm chooses the candidate patch that returns the lowest value from the cost function; in the case of multiple patches with the same value, the system selects randomly from them.

The cost function itself attempts to minimise the visual error between placed patches, using two properties of the roads which lie along the seam: their location and size. Specifically, the function was designed to return a higher error cost when the roads on each side of the seam are further apart, and when they are of different sizes. The function is evaluated as:

$$cost(a,b) = \sum_{i=1}^{numRoads} (compat(a_i, b_i) * a_{i_{size}} * b_{i_{size}}) \quad (1)$$

where

$$dist(r1, r2) = |r1 - r2|^2 \quad (2)$$

$$ratio(s1, s2) = max(s1, s2)/min(s1, s2) \quad (3)$$

$$compat(a,b) = dist(a_{pos}, b_{pos}) * ratio(a_{size}, b_{size}) \quad (4)$$

It is much more visually jarring in a synthesised map when a major road is disconnected from another along a seam than when the same problem occurs with a side road or pathway. For this reason, each road pair cost is also multiplied by the product of their sizes. This results in a lower contribution to the cost total by smaller roads, and thus disconnects on major roads are discouraged more strongly.

The patches are therefore chosen as:

$$\arg\min_p \ cost(patch(x-1, y), p) \quad (5)$$

for top row patches,

$$\arg\min_p \ cost(patch(x, y-1), p) \quad (6)$$

for leftmost column patches, and

$$\arg\min_p \ (cost(patch(x-1, y), p) + cost(patch(x, y-1), p)) \quad (7)$$

for the remaining patch spaces.

For comparison purposes, each of the fragments stored by the lookup table is used as a template. On choosing a fragment to fill a space within the grid, the path and building data are copied to a new patch, so that the path and building data may be modified by further steps within the algorithm, without corrupting the patch data stored in the lookup table.

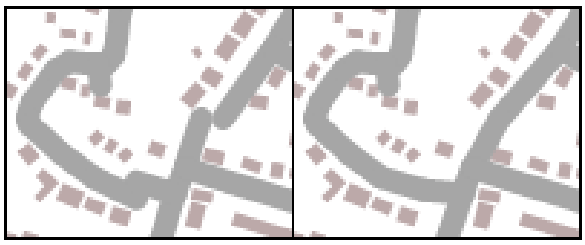
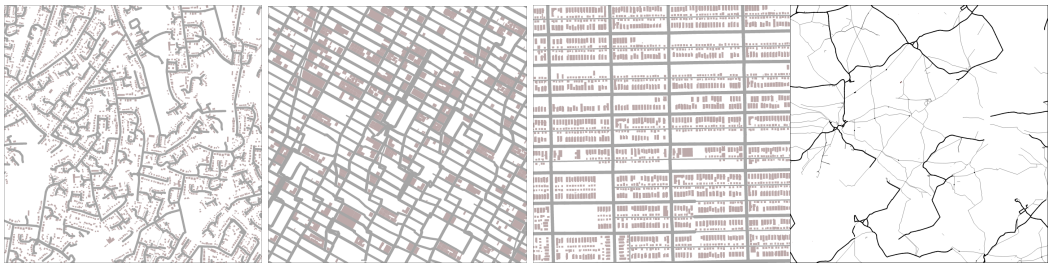


Figure 2: Fix step applied to disconnected roads. Before (left) and after (right).

Figure 3: Fix step applied to floating roads. Before (left) and after (right).

The next stage of the process is to deal with road disconnects along patch seams (Figure 2) and floating roads that are fully disconnected (Figure 3). To remove disconnection of roads along patch seams, the endpoint of

Figure 4: Test inputs. Left to right: Chicago, IL; Southampton, UK; Northampton UK; Times Square, NY.



(a) (b) (c) (d)

Figure 5: Cities generated from a single input map. a) Southampton. b) Times Square. c) Chicago. d) Northampton.

each road on a given patch edge is translated by half of the distance to the corresponding endpoint in the adjacent patch. This results in a join between the roads, at the cost of introducing some heuristic modification to the road network. The distance between corresponding road ends is usually small enough that these modifications are hard to detect in all but the most regularly structured maps. This issue is discussed further in the results section.

4 Results and Evaluation

Four test maps (Figure 4) were used to evaluate PatchCity. Together they represent four of the most common styles of road network: (i) irregular city roads with varying shapes and directions (Southampton); (ii) irregular, grid-aligned roads (New York); (iii) regular, grid-aligned roads with high building density (Chicago); and (iv) sparse, disorganised country roads (Northampton).

Plausibly realistic cities are produced (Figure 5). There are, however, some issues: (i) removal of floating road sections can produce large holes, (ii) buildings that cross patch boundaries are excluded, and (iii) cities with a grid structure can show a visible seam.

Floating roads start and end within the boundaries of the map area. The removal of these problematic roads is implemented by identifying the connected component subgraphs of the road network. Subgraphs that have no roads leading off any of the boundaries of the map are identified as floating roads. When floating roads are removed, buildings may become detached and so also need to be removed. This is achieved by creating a map which associates each building with the nearest path node. After removing a floating road, any building which maps to those paths is also removed. The removal of floating roads and their accompanying buildings can introduce noticeable holes within the map. Consequently this stage is optional as it may or may not improve the synthesised results. The algorithm could be improved with an additional step that connects floating road subgraphs to the rest of the map by inserting an additional connecting road.

In addition, during the population of the lookup table, extracted patches contain only those buildings that fall entirely within their boundaries. This is problematic when buildings are either too large to fit within the specified patch size, or large enough that they are excluded from most patches in their area. This results in maps which often contain large gaps in building density along seam lines (Figure 6). This issue could be addressed

by including the clipped buildings within a patch, and only inserting them into the final synthesised map if this does not result in a collision with neighbouring patch elements.

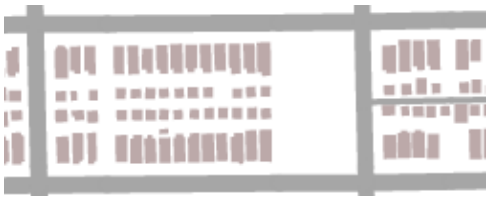


Figure 6: Example of building placement gaps occurring due to patch cropping.

A noticeable artifact within the synthesised Times Square result is that at scattered points around the map the road network falls out of line with its intended grid structure, creating a knock on effect for the subsequent patches (Figure 7). This problem exists largely because PatchCity operates on a fixed grid of lookup table patch locations, which may not align precisely with the city's grid structure. These artifacts could be reduced by searching in the local neighbourhood of lookup table patches prior to placement to find the best possible match between placed tiles.

Figure 7: Example of the synthesis method failing to fully preserve grid-based road alignment.

Despite these issues, grid network generation is still possible with the algorithm if the input road network has regularity, i.e. blocks within the input grid network are regularly spaced, as shown in the Chicago synthesis example (Figure 5 (a)).

Although no explicit optimisation has been performed, PatchCity's main synthesis algorithm takes place in real-time. The execution time for the selection of patches is proportional to the number of lookup table entries. In addition, the removal of floating roads requires a calculation of the connected components which is linear in the number of road segments and junctions. This implementation's measured performance is linear in the area of the output map. The ability to generate in real-time gives PatchCity a much wider range of applications. For example, its use can be easily imagined in an open world video game which generates the environment around the player as they travel, with the potential for 'infinite' worlds.

5 Conclusions and Further Work

The key strength of PatchCity is its use of structural example data as input. This enables the algorithm to generate a diverse range of city styles, from regular, grid-aligned modern cities to more organic, historic urban centers. PatchCity is also able to capture instances of uncommon city features such as shortcut pathways and roundabouts. This adds to the realism and variety of the output.

PatchCity's building synthesis is a result of the algorithm's extraction of building data from the input maps, providing building placement as a built-in element of its execution. As further work, additional city metadata could be included in the synthesis process. Terrain data, land use models, height maps and population density information could all be added to the input and integrated into the patch extraction and placement process. By adding more terms to the cost function, this additional data could provide more detailed and realistic city structures.

Further improvements in city quality could also be achieved by performing user evaluations of synthetic results. With such evaluations it may be possible to develop improved metrics for tile placement as well as

a more global method of ranking different synthesised cities. Knowledge of what constitutes a high quality output would allow the system to repeat the synthesis process until the quality reaches a specified threshold.

The PatchCity algorithm can also be viewed as a novel contribution to the field of texture synthesis. Highly stylised structural data often produces artifacts using traditional pixel based methods. By adapting the cost function described in Section 3.1, PatchCity could be generalised to other synthesis applications.

References

[Aliaga et al., 2008] Aliaga, D. G., Vanegas, C. A., and Beneš, B. (2008). Interactive example-based urban layout synthesis. In *ACM Transactions on Graphics (TOG)*, volume 27, page 160. ACM.

[Chen et al., 2008] Chen, G., Esch, G., Wonka, P., Müller, P., and Zhang, E. (2008). Interactive procedural street modeling. *ACM Transactions on Graphics (TOG)*, 27(3):103.

[Efros and Freeman, 2001] Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM.

[Efros and Leung, 1999] Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038. IEEE.

[Groenewegen et al., 2009] Groenewegen, S. A., Smelik, R. M., de Kraker, K. J., and Bidarra, R. (2009). Procedural city layout generation based on urban land use models. In *Proceedings of the 30th Annual Conference of the European Association for Computer Graphics (EurographicsâĂŹ09)*, pages 45–48.

[Haklay and Weber, 2008] Haklay, M. M. and Weber, P. (2008). Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18.

[IDV Inc., 2014] IDV Inc. (2014). Speedtree. http://www.speedtree.com.

[Kwatra et al., 2003] Kwatra, V., Schödl, A., Essa, I., Turk, G., and Bobick, A. (2003). Graphcut textures: image and video synthesis using graph cuts. In *ACM Transactions on Graphics (ToG)*, volume 22, pages 277–286. ACM.

[Lechner et al., 2004] Lechner, T., Watson, B., Ren, P., Wilensky, U., Tisue, S., and Felsen, M. (2004). Procedural modeling of land use in cities.

[Lechner et al., 2003] Lechner, T., Watson, B., and Wilensky, U. (2003). Procedural city modeling. In *In 1st Midwestern Graphics Conference*.

[Parish and Müller, 2001] Parish, Y. I. and Müller, P. (2001). Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM.

[Planetside Software, 2014] Planetside Software (2014). Terragen. http://www.planetside.co.uk.

[Rozenberg and Salomaa, 1980] Rozenberg, G. and Salomaa, A. (1980). *Mathematical Theory of L Systems*. Academic Press, Inc.

[Sony Online Entertainment, 2014] Sony Online Entertainment (2014). Everquest Next. http://www.everquestnext.com.

[Sproull and Newman, 1973] Sproull, B. and Newman, W. M. (1973). *Principles of Interactive Computer Graphics*. McGraw-Hill Education, international edition.